An Overview of C

Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Spring 2023

What is Programming?

Computer Systems











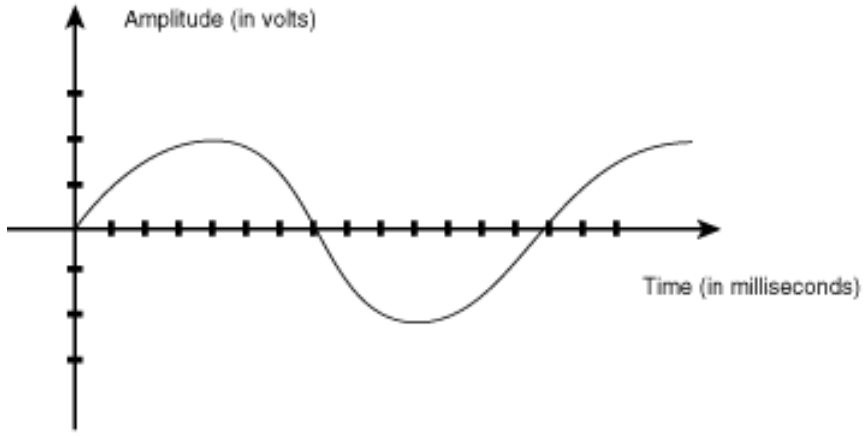


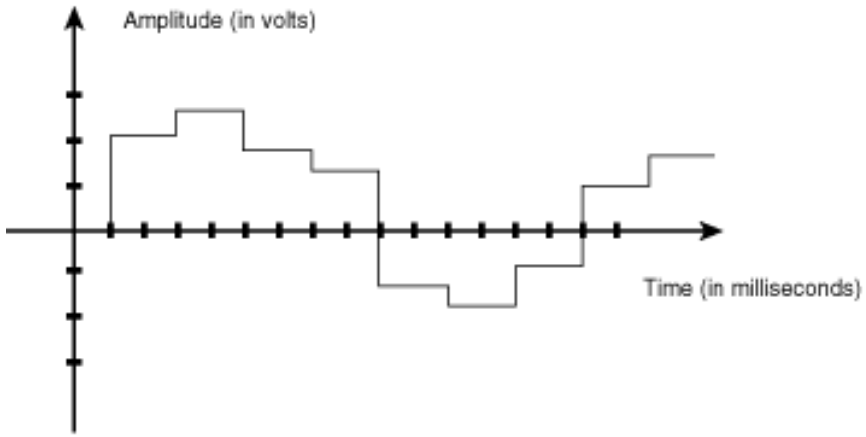

What is a Computer?

A computer is a  machine.

What is a Digital Computer?

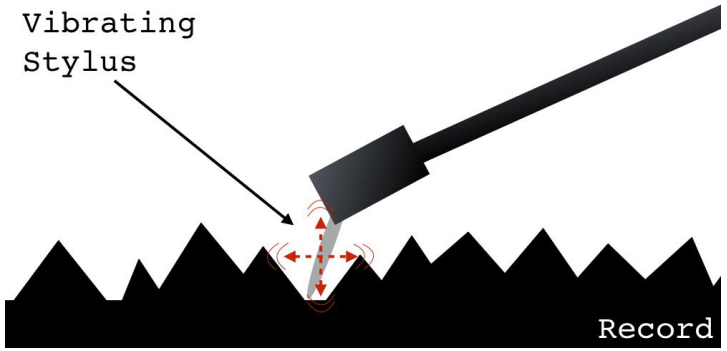
- Analog vs. digital?

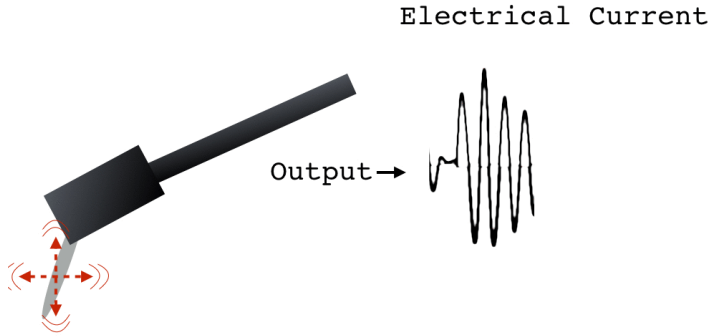


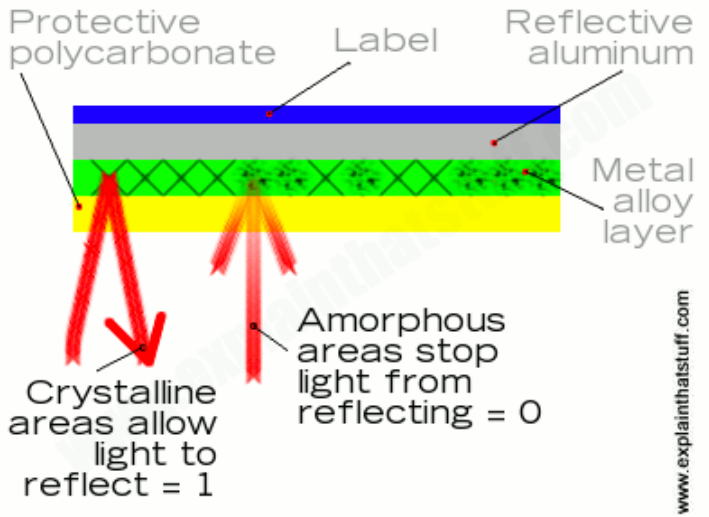


- Compact Disc (CD)
  - 44.1 KHz, 16-bit, 2-channel
- MP3
  - A digital audio encoding with lossy data compression

Example: LP Record vs. Compact Disc

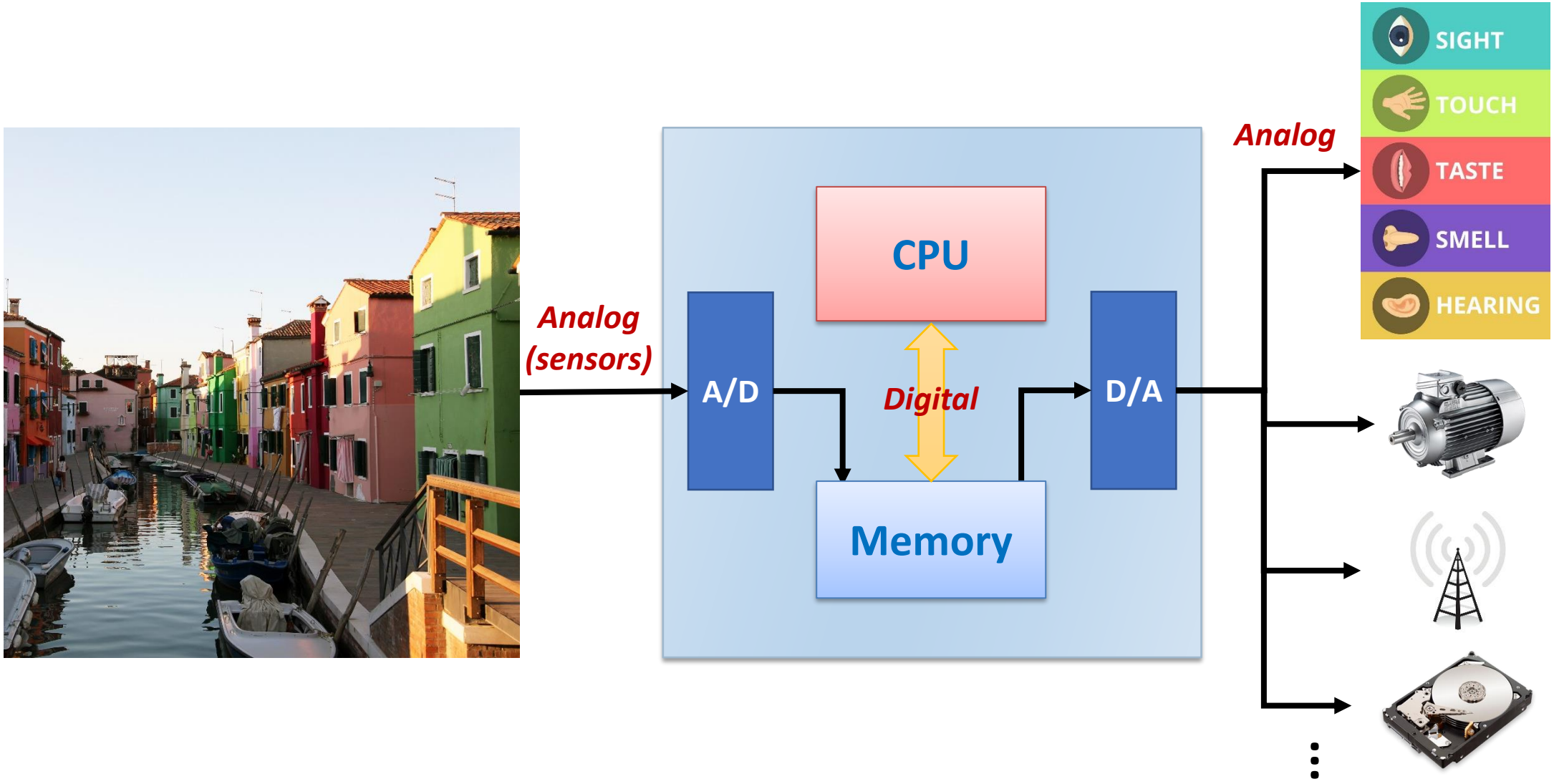






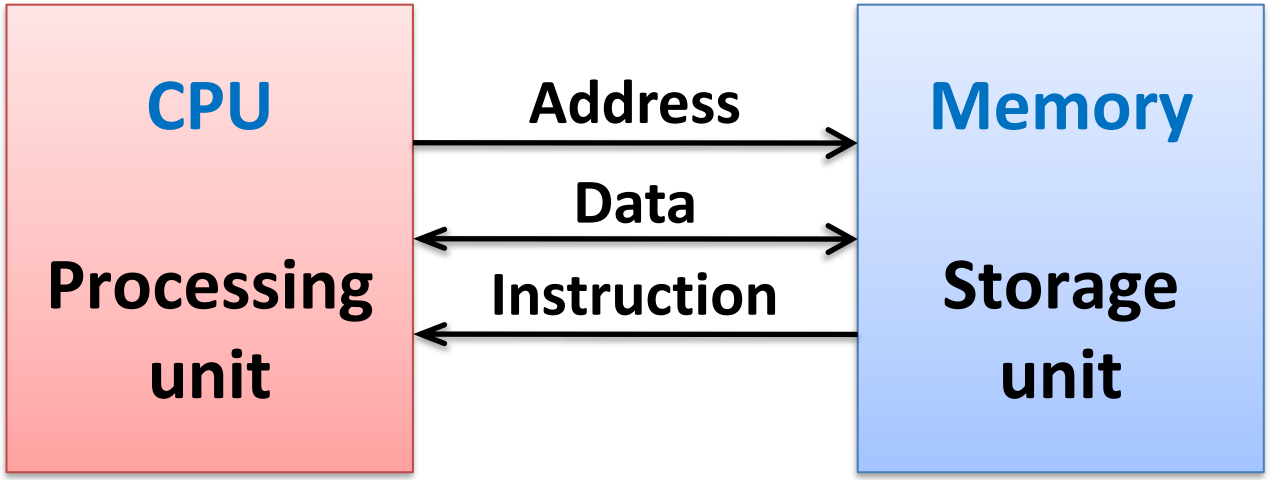
Source: http://www.soundsetal.com/blog-how-do-vinyl-records-work/
http://www.explainthatstuff.com/cdplayers.html

Digital Computer



Von Neumann Architecture

- By John von Neumann, 1945



Data movement
Arithmetic & logical ops
Control transfer

Byte addressable array
Code + data (user program, OS)
Stack to support procedures

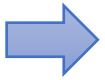
A Case for Robot Vacuum Cleaner

- Commands (or instructions)
  - FWD: Go one step forward `00`
  - BWD: Go one step backward `01`
  - LEFT: Turn left `10`
  - RIGHT: Turn right `11`

- A simple program

```
FWD 2;
RIGHT;
FWD 1;
RIGHT;
```

```
FWD 2;
LEFT;
FWD 1;
LEFT;
```



```
00 00 11 00 11
00 00 10 00 10
00 00 11 00 11
00 00 10 00 10  ...
```

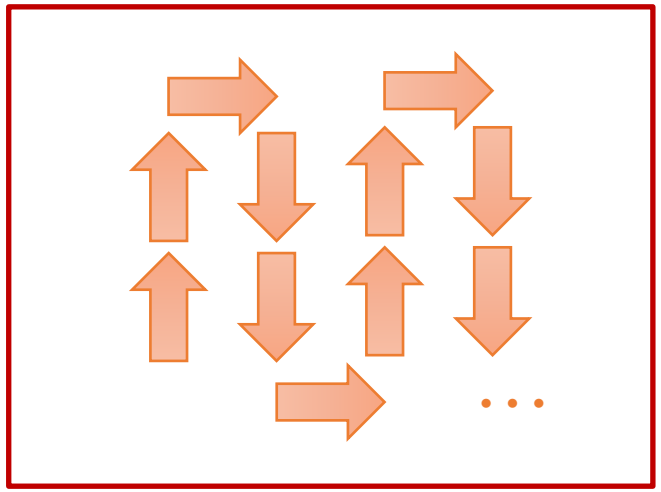
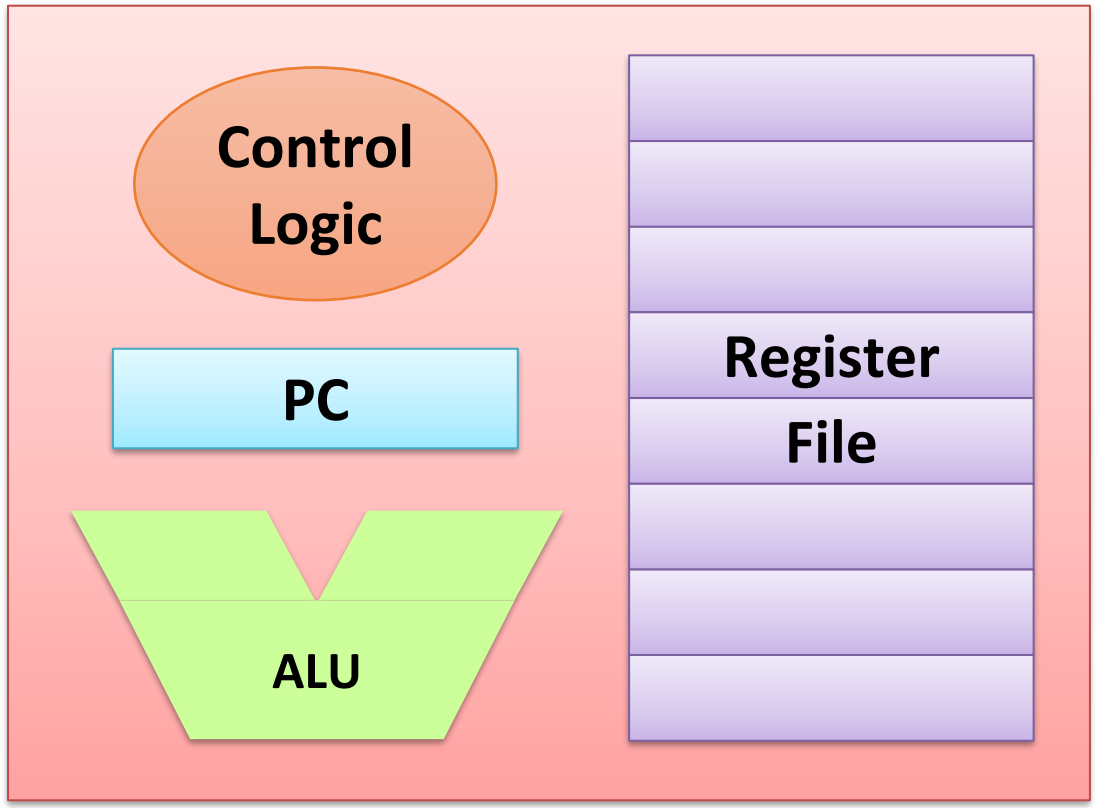


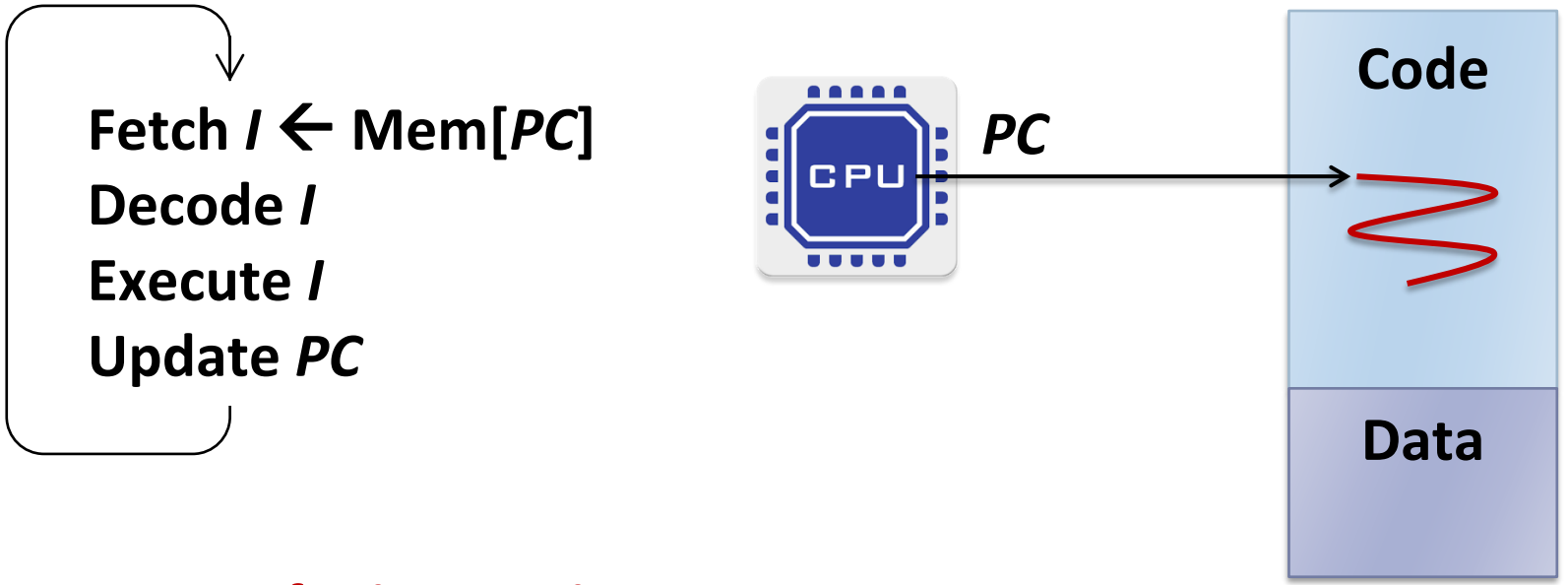
Image from https://www.lge.co.kr/vacuum-cleaners

CPU

- Central Processing Unit
  - PC (Program Counter)
    - Address of next instruction
  - Register file
    - Heavily used program data
  - ALU (Arithmetic & Logic Unit)
    - Arithmetic operations
    - Logical operations
  - Control logic
    - Control instruction fetch, decoding and execution



The (Dumb) Life of CPU



Data transfer instructions
Arithmetic/Logical instructions
Control transfer instructions

Programming Elements: Data Types

- Integer: 8-bit, 16-bit, 32-bit, 64-bit
  - Unsigned
  - Signed (Two's complement)

- Floating point
  - Single precision: 32-bit
  - Double precision: 64-bit

Programming Elements: Operations

- Computation
  - Arithmetic operations: Addition, Subtraction, Multiplication, Division, …
  - Logical operations: AND, OR, XOR, NOT,
    Logical shift left/right, Arithmetic shift/right, …
- Memory access: Load & Store
- Conditional branch
  - Conditional: IF … THEN … ELSE
  - Loops: FOR, WHILE, …
- All problems that can be solved by programming can be decomposed into these elements (+ I/O)

Programming 101

- What data do I need?

- How to organize data?

  or What "data structure" do I need?

- What operations should be made to data?

  or What "algorithm" should I use?

Our First C Program

helloworld.c (1)

- Create a C source file (*.c)
  - Use a text editor (e.g., vi, vim, emacs, notepad, ...) or
  - Use an IDE (e.g., Microsoft Visual Studio Code, Eclipse, …)

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
    return 0;
}
```

helloworld.c (2)

- Compile and linking
  - Convert source code to executable file
  - Compiler and linker do the job
- Run the program

```
$ gcc helloworld.c
$ ls -l
total 12
-rwxrwxrwx 1 jinsoo jinsoo 8304 Mar 5 20:16 a.out
-rwxrwxrwx 1 jinsoo jinsoo   77 Mar 5 20:14 helloworld.c
$ ./a.out
hello, world
```

Going Deeper into C (1)

```c
#include <stdio.h>

int main(void)
{

    printf("hello, world\n");

    return 0;

}
```

- Preprocessor
  - Built into the C compiler
  - Lines beginning with #: communicate with the preprocessor
- `#include`
  - Preprocessor includes a copy of the header file `stdio.h`
- `stdio.h`
  - Provided by the C system
  - Declaration of standard input/output functions such as `printf()`

Going Deeper into C (2)

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");

    return 0;
}
```

- Function definition for `main()`
  - Every program has a function named `main()`
  - `main()` is the starting point of the program
  - `int`, `void`: keywords (special meaning to the compiler)
  - `int`: returns an integer value
  - `void`: no argument
  - { ... }: the body of the function

Going Deeper into C (3)

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");

    return 0;

}
```

- Statement
  - Ends with a semicolon
- `printf()`
  - A function that prints on the screen
  - Information in the header file `stdio.h`
  - Part of the standard C library
- `"hello, world\n"`
  - `" ... "`: string constant in C
  - `\n`: a single character called newline

Going Deeper into C (4)

```c
#include <stdio.h>

int main(void)
{

    printf("hello, world\n");

    return 0;

}
```

- `return 0;`
  - A return statement
  - Causes the value *zero* to be returned to the operating system (or to the calling function)

Compiling

- Convert source file to object file
  - `helloworld.c` to `helloworld.o`

- Object file
  - A file with expressions that computers can understand

- When compiling fails?
  - Something wrong with the source file
  - Syntax error: expressions with wrong C grammar

Compile Errors

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
    return 0;                          /* wrong!!! */
}
```

```
$ gcc helloworld.c
helloworld.c: In function ‘main’:
helloworld.c:6:2: error: ‘return’ undeclared (first use in this function)
return 0;
^~~~~~
helloworld.c:6:2: note: each undeclared identifier is reported only once for each
function it appears in
helloworld.c:6:9: error: expected ‘;’ before numeric constant
return 0;
^
$
```

Linking and Running a Program

- Linking
  - The process to make an executable program out of object file(s)
  - Default executable file name: `a.out` (Use `-o` to specify the file name)

- Running a program
  - Type the executable file name

```
$ gcc -g -o helloworld helloworld.c
$ ./helloworld
hello, world
```

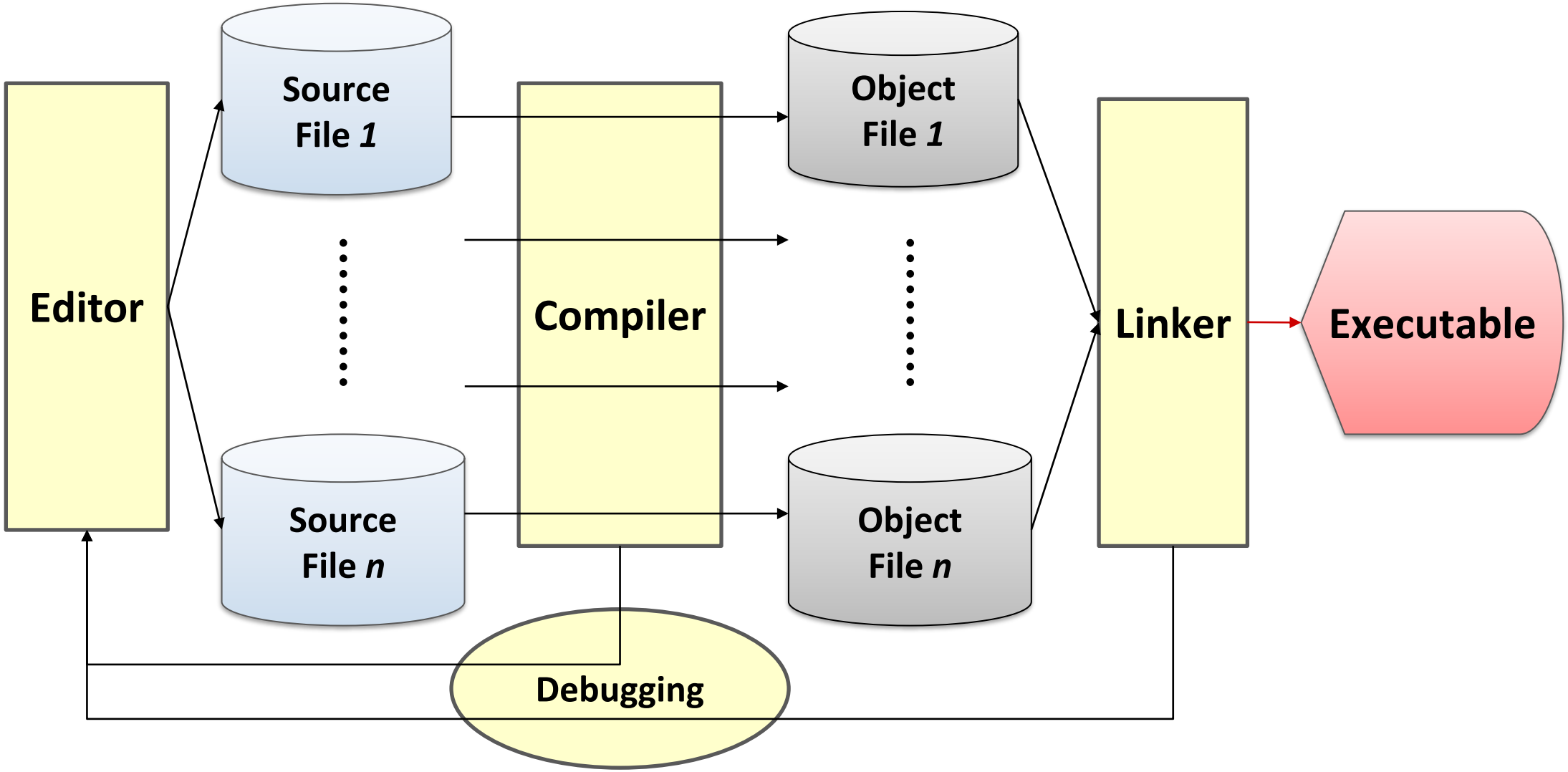
Debugging

- Debugging
  - Fix the bugs in the source code
  - Debugger (e.g., gdb) does the job

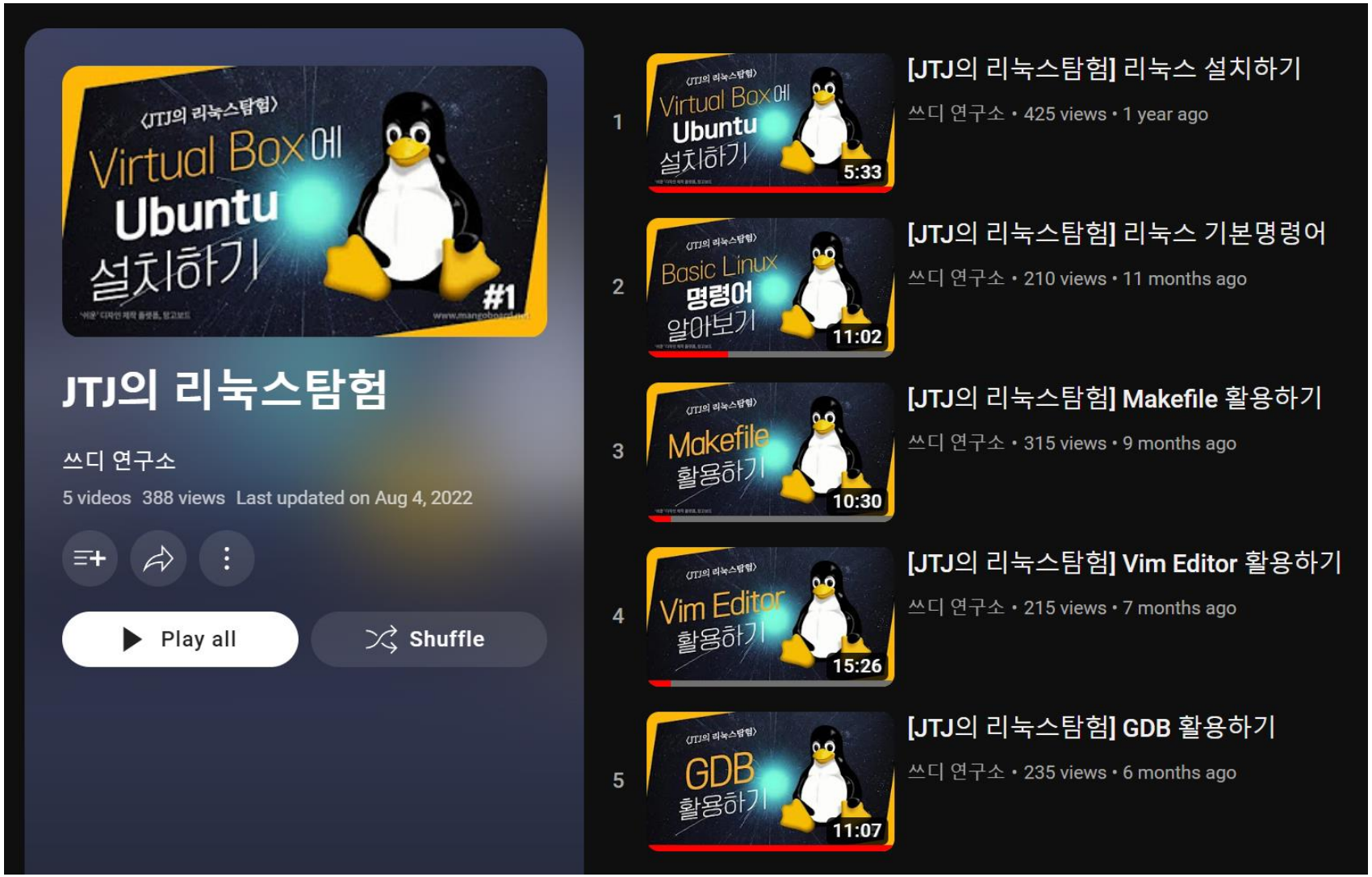
  - Breakpoints
  - Watchpoints
  - Single stepping
  - Examining variables
  - Examining memory
  - ...

```
$ gdb helloworld
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from helloworld...done.
(gdb) list
1 #include <stdio.h>
2
3 int main(void)
4 {
5 printf("hello, world\n");
6 return 0;
7 }
(gdb) run
Starting program: /home/jinsoo/pp/helloworld
hello, world
[Inferior 1 (process 258) exited normally]
(gdb)
```

From Source to Executable



FYI:


〈JTJ의 리눅스탐험〉
Virtual Box에 Ubuntu 설치하기
#1
JTJ의 리눅스탐험
쓰디 연구소
5 videos 388 views Last updated on Aug 4, 2022
Play all
Shuffle
1
Virtual Box에 Ubuntu 설치하기
5:33
[JTJ의 리눅스탐험] 리눅스 설치하기
쓰디 연구소 • 425 views • 1 year ago
2
Basic Linux 명령어 알아보기
11:02
[JTJ의 리눅스탐험] 리눅스 기본명령어
쓰디 연구소 • 210 views • 11 months ago
3
Makefile 활용하기
10:30
[JTJ의 리눅스탐험] Makefile 활용하기
쓰디 연구소 • 315 views • 9 months ago
4
Vim Editor 활용하기
15:26
[JTJ의 리눅스탐험] Vim Editor 활용하기
쓰디 연구소 • 215 views • 7 months ago
5
GDB 활용하기
11:07
[JTJ의 리눅스탐험] GDB 활용하기
쓰디 연구소 • 235 views • 6 months ago

Variables, Expressions, and Assignments

Distance of a Marathon in Kilometers

- Marathon: 26 miles 385 yards
- 1 yard → 1/1760 mile
- 1 mile → 1.609 km

Marathon

- (26 + 385/1760) miles
- (26 + 385/1760) x 1.609 km

```
m = 26
y = 385
km = (m + y/1760) x 1.609
```

marathon.c

```c
/* the distance of a marathon in kilometers */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is %f kilometers.\n", kilometers);
    return 0;
}
```

marathon.c

```c
/* the distance of a marathon in kilometers */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is %f kilometers.\n", kilometers);
    return 0;
}
```

- `/* ... */`
  - **Comment**
  - **Ignored by the compiler**

marathon.c

```c
/* the distance of a marathon in kilometers */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is %f kilometers.\n", kilometers);
    return 0;
}
```

- **int**
  - A keyword, integer value
  - Declaration of the variables **miles** and **yards** of type **int**
  - Declarations and statements end with a semicolon
- **Variable**
  - A memory space to hold a value

marathon.c

```c
/* the distance of a marathon in kilometers */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is %f kilometers.\n", kilometers);
    return 0;
}
```

- `float`
  - A keyword, floating-point value
  - Declaration of the variable `kilometers` of type `float`

marathon.c

```c
/* the distance of a marathon in kilometers */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is %f kilometers.\n", kilometers);
    return 0;
}
```

- **Assignment statement**
  - variable = expression
  - Equal sign (=): assignment operator
  - The value of the expression on the right side of the equal sign is assigned to the variable

marathon.c

```c
/* the distance of a marathon in kilometers */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is %f kilometers.\n", kilometers);
    return 0;
}
```

- **Expression**
  - **On the right side of assignment operator**
  - **Constants, variables, or combinations of operators with variables and constants**

marathon.c

```c
/* the distance of a marathon in kilometers */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is %f kilometers.\n", kilometers);
    return 0;
}
```

- **Integer constants**
  - **Integer types: `char`, `short`, `int`, `long`, …**

- **Floating-point constants**
  - `float`, `double`, …
  - **Floating-point constants are automatically of type `double`**

marathon.c

```c
/* the distance of a marathon in kilometers */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is %f kilometers.\n", kilometers);
    return 0;
}
```

- **Evaluation of expression**
  - **Division of two integers results in an integer value (i.e., 7/2 is 3)**
  - **A double divided by an integer: Integer is automatically converted to double (i.e., 7.0 / 2 is 3.5)**
  - **1.609 * (miles + yards / 1760) → BUG!!!**

marathon.c

```c
/* the distance of a marathon in kilometers */

#include <stdio.h>

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is %f kilometers.\n", kilometers);
    return 0;
}
```

```
A marathon is 42.185970 kilometers.
```

- **String formatting**
  - `%f`: **format, conversion specification**
  - **Matched with the remaining argument, the variable `kilometers`**

Better marathon.c

```c
/* the distance of a marathon in kilometers */
#include <stdio.h>

#define KM_PER_MILE     1.609
#define YARDS_PER_MILE  1760.0

int main(void)
{
    int     miles, yards;
    float   kilometers;

    miles = 26;
    yards = 385;
    kilometers = KM_PER_MILE * (miles + yards / YARDS_PER_MILE);
    printf("A marathon is %f kilometers.\n", kilometers);
    return 0;
}
```

avgscore1.c

```c
#include <stdio.h>

int main(void)
{
    int score1, score2, score3, avg_score;
    int num_score;

    score1 = 87;
    score2 = 93;
    score3 = 100;
    num_score = 3;
    avg_score = (score1 + score2 + score3) / num_score;
    printf("Average score: %d\n", avg_score);
    return 0;
}
```

```
Average score: 93
```

avgscore2.c

```c
#include <stdio.h>

int main(void)
{
    float fscore1, fscore2, fscore3, avg_fscore;
    int num_score;

    fscore1 = 87.0;
    fscore2 = 93.0;
    fscore3 = 100.0;
    num_score = 3;
    avg_fscore = (fscore1 + fscore2 + fscore3) / num_score;
    printf("Average score: %f\n", avg_fscore);
    return 0;
}
```

```
Average score: 93.333336
```

Why not 93.333333?

Flow of Control

`if` Statement

```c
#include <stdio.h>

int main(void)
{
    int a, b;

    a = 1;

    if (b == 3)
        a = 5;

    printf("a is %d\n", a);
    return 0;
}
```

- `if (`*expr*`)` *statement*
  - If *expr* is nonzero (true), then *statement* is executed
  - Otherwise, it is skipped
- `b == 3`
  - `==`: Equal operator
  - Logical expression -- evaluated to either the integer value 1 (true) or 0 (false)

if Statement: Examples

```c
#include <stdio.h>

int main(void)
{
    int a, b;

    b = 3;
    a = 1;
    if (b == 3)
        a = 5;
    printf("a is %d\n", a);
    return 0;
}
```

```
a is 5
```

```c
#include <stdio.h>

int main(void)
{
    int a, b;

    b = 2;
    a = 1;
    if (b == 3)
        a = 5;
    printf("a is %d\n", a);
    return 0;
}
```

```
a is 1
```

if/else

```c
#include <stdio.h>

int main(void)
{
    float score = 82.0;
    char  grade;

    if (score >= 40.0)
    {
        grade = 'S';
        printf("Pass (%c)\n", grade);
    }
    else
    {
        grade = 'U';
        printf("Fail (%c)\n", grade);
    }
    return 0;
}
```

- `if (`*expr*`)` *S1* `else` *S2*
  - If *expr* is nonzero (true), then *S1* is executed
  - Otherwise, *S2* is executed
- Compound statement
  - A group of statement surrounded by braces
  - A statement, itself
  - No semicolon needed at the end

if/else if/else

```c
#include <stdio.h>

int main(void)
{
    float score = 83.5;
    char  grade;

    if (score >= 90.0)
        grade = 'A';
    else if (score >= 80.0)
        grade = 'B';
    else if (score >= 60.0)
        grade = 'C';
    else if (score >= 40.0)
        grade = 'D';
    else
        grade = 'F';
    printf("Your grade is %c\ns", grade);
    return 0;
}
```

- The test order matters!

```c
    /* ??? */

    if (score >= 80.0)
        grade = 'B';
    else if (score >= 90.0)
        grade = 'A';
    else ...
```

while Statement

```c
#include <stdio.h>

int main(void)
{
    int i = 1;
    int sum = 0;

    while (i <= 5)
    {
        sum = sum + i;
        i++;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```

- `while (`*`expr`*`)` *`statement`*
  - If *`expr`* is true, the (compound) *`statement`* is executed
  - The control is then passed back to the beginning of the while loop for the process to start over again
  - The while loop is repeatedly executed until the test fails
- `i++`
  - `++`: Increment operator
  - `i = i + 1`

for Statement

```c
#include <stdio.h>

int main(void)
{
    int i;
    int sum = 0;

    for (i = 1; i <= 5; i++)
    {
        sum = sum + i;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```

- `for` (*init*; *test*; *update*) *statement*;
  - Semantically equivalent to:

```c
init;
while (test)
{
    statement;
    update;
}
```

Function

- A piece of code that is a building block in the problem-solving process
- A C program consists of one or more functions

```c
#include <stdio.h>

int min(int x, int y)
{
    if (x < y)  return x;
    else        return y;
}

int max(int x, int y)
{
    if (x > y)  return x;
    else        return y;
}
```

```c
int main(void)
{
    int a, b;

    scanf("%d %d", &a, &b);

    printf("max(%d, %d) = %d\n",
            a, b, max(a, b));
    printf("min(%d, %d) = %d\n",
            a, b, min(a, b));
    return 0;
}
```

Summary: C Program is ...

- A sequence of functions
  - `main()` function is executed first
- A function definition has a form:

```
type function_name(parameter_list)
{
    declarations
    statements
}
```

- Declarations:
  - Variable names and their types
- Statements:
  - Computation or control

```
    int   miles = 26;
    float km;
```

```
    km = miles * 1.609;
    printf("%f", km);
    if (cond) { ... }
    while (cond) { ... }
```